# Facilitating Requirements Inspection with Search-Based Selection of Diverse Use Case Scenarios

Huihui Zhang
Beihang University
Beijing, China
zhhui@buaa.edu.cn

Tao Yue
Simula Research Laboratory and University of Oslo
Oslo, Norway
tao@simula.no

Shaukat Ali
Simula Research Laboratory
Oslo, Norway
shaukat@simula.no

Chao Liu
Beihang University
Bejing, China
liuchao@buaa.edu.cn

## ABSTRACT

Use case scenarios are often used for conducting requirements inspection and other relevant downstream activities. While working with industrial partners, we discovered that an automated solution is required for optimally selecting a subset of use case scenarios, aiming to enable cost-effective requirements inspection. In this paper, relying on a natural language based use case modeling methodology to specify requirements as use case models and derive use case scenarios automatically, we propose a search based and similarity function based approach to optimally select most diverse use case scenarios from the ones automatically generated from the use case models. We conducted an empirical study to evaluate the performance of various search algorithms together with eight similarity functions, through an industrial case study and six case studies from the literature. Results show that the search algorithms significantly outperformed Random Search and (1+1) Evolutionary Algorithm together with the Normalized Longest Common Subsequence (NLCS) similarity function performed significantly better than the other 31 combinations of the search algorithms and similarity functions for most of the problems.

## Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications–*Methodologies and Tools.*

## General Terms

Algorithms, Measurement, Performance, Experimentation.

## Keywords

Use Case Inspection, Scenarios Selection, Search Algorithms, Similarity Functions, Empirical Study.

## 1. INTRODUCTION

Requirements play a critical role in the development of any non-trivial software system [15]. Use case modeling is one of the most widely used requirements specification techniques. By combining diagrammatic and textual descriptions, use case models offer an intuitive and precise foundation for requirements specification. The essential part of a use case is its scenarios, each of which describes a sequence of steps/actions that are executed under a specific set of conditions.

In a practical context, a use case can lead to the generation of a non-trivial number of scenarios, especially when the use case incudes or is extended by one or more other use cases. Use case scenarios are often the input for conducting requirements inspection and analysis, supporting requirements-based testing and other relevant downstream activities [14]. Manually requirements inspection requires domain experts walking through all available scenarios, identify defects and fix them if there exists any [10]. It is often impossible to walk through all scenarios. Hence, a common practice is to select a subset of the scenarios to inspect within limited time and human resources. The selection criteria are either based on domain expert's tacit knowledge or simply random [10].

In this paper, we propose an automated, systematic and similarity-based approach to identify an optimal subset of scenarios for manual inspection, using search algorithms. The objective is to maximize the diversity of selected scenarios, expecting that more defects can be identified during a requirements inspection process. Notice that different scenarios of a given use case might contain common steps since they are derived from the same use case. It is therefore important to select scenarios that cover as many different steps as possible.

Search algorithms with similarity functions have already been adopted to address optimization problems regarding test case selection or prioritization (e.g., [13]). However, to the best of our knowledge, there is no work focusing on using similarity-based techniques combined with search algorithms to address the use case scenario selection problem. In this paper, we propose and assess a fitness function for addressing this optimization problem. We evaluate the fitness function in conjunction with the following search algorithms, i.e., Steady State Genetic Algorithms (SSGA) [26], (1+1) Evolutionary Algorithm ((1+1)EA) [22], Alternating Variable Method (AVM) [27]. Random Search (RS) was used as the baseline to evaluate the performance of these algorithms. To determine the diversity of scenarios, we evaluated eight similarity functions: Counting function (CNT) [5], Jaccard Index (JAC) [4], Gower-Legendre (GOW) [4], Sokal-Sneath (SOK) [4], Normalized Longest Common Subsequence (NLCS), Levenshtein Distance (LEV) [6], Needleman-Wunsch (NW) [7] and Smith-Waterman (SW) [7].

One industrial case study from the avionics domain and six carefully selected case studies from the literature, with in total 38 use case specifications and 1667 scenarios were used to evaluate the performance of the selected search algorithms and similarity functions. Results show that all the search algorithms significantly outperformed RS, indicating the usefulness of applying search.

The rest of the paper is organized as follows. Section 2 describes key techniques of our methodology including: the use case modeling approach, automated generation of use case scenarios, selected similarity measurements and search algorithms. In Section 3, we specify the optimization problem and present the

BICT 2015, December 03-05, New York City, United States
Copyright © 2016 ICST
DOI 10.4108/eai.3-12-2015.2262435

fitness functions. Empirical evaluation is reported in Section 4. Section 5 addresses the threats to validity. Related work is provided in Section 6 and we conclude the paper in Section 7.

## 2. BACKGROUND

We describe the use case modeling methodology in Section 2.1, followed by the similarity functions in Section 2.2.

### 2.1 Use Case Modeling and Scenario Derivation

In [15], a restricted use case modeling methodology (RUCM), which encompasses a use case template and a set of restriction rules for the textual use case specifications, was proposed. A use case specification has one basic flow and zero to many alternative flows. An alternative flow always depends on a condition in a reference flow, which can be the basic flow or another alternative flow. There are three types of alternative: A specific/bounded/global alternative flow refers to a specific step/more than one steps/any step in the reference flow. This methodology is generic and can be extended for other purposes such as specifying and generating test cases [19]. RUCM includes a use case metamodel [25] to formalize use cases with the objective to enable automated analysis.

In RUCM, a use case scenario is a sequence of steps (sentences) that only forms one branch to execute. RUCM captures control flow information in a structured way through flows of events and keywords (e.g., DO-UNTIL and INCLUDE USE CASE). Based on the control flow information embedded in use case models, RUCM supports the following three coverage criteria: *All Condition Coverage*, *All FlowOfEvents Coverage* and *All Sentence Coverage* to automatically generate use case scenarios.

The *All Condition Coverage* criterion ensures that all conditions are covered at least once. The *Loop Coverage* criterion ensures that each loop is exercised *x* number of times, where *x* can be specified by a user. In the current implementation of the *Loop Coverage* criterion, each loop is exercised exactly once. The *All FlowOfEvents Coverage* criterion makes sure that the basic and all the alternative flows of a use case specification are covered at least once. The *All Sentence Coverage* criterion generates a set of scenarios that cover all the sentences of a use case specification at least once. Note that the scenario derivation approach has been applied to automatically generate test cases [19].

### 2.2 Similarity Measures

#### 2.2.1 Set-Based Similarity Measures

Set-based similarity functions are widely used in data mining, in our context, each use case scenario is a vector of elements and each element is a step (sentence). However, the vector size can be different since the length of scenarios (i.e., the number of steps) may vary. As an example, taking two scenarios *sn1* = {*s1*, *s2*, *s3*, *s5*, *s6*, *s3*} and *sn2* = {*s1*, *s2*, *s4*, *s6*} as inputs, in our context, *s1*-*s6* are different steps of the flows of events of a particular use case specification. Scenarios *sn1* and *sn2* share the common steps: *s1*, *s2* and *s6*. We used four set-based similarity functions in our experiments: CNT [5], JAC [4], GOW [4], and SOK [4].

*Counting function (CNT).* The Counting function is borrowed from [5] for comparing two sets of transitions in a specific modeling language. We have defined a generalized version of this function as the number of identical elements in the input sets divided by the average size of inputs. The formula for calculating similarity of two scenarios (denoted A and B) is: $CNT(A,B) = \frac{(|A\cap B|)}{(sizeof(A)+sizeof(B))/2}$ (1), where |A ∩ B| is the size of intersection of A and B (i.e., common steps of the two scenarios) and the function *sizeof()* calculates the element size of a set (i.e., number of the steps of a scenario). For example, when taking *sn1* and *sn2* as the input, CNT(sn1, sn2) = 3 / ((5+4) / 2) = 0.6667.

*Jaccard Index (JAC).* In [4], JAC is defined with formula as: $JAC(A,B) = \frac{|A\cap B|}{|A\cup B|} = \frac{|A\cap B|}{|A\cap B|+1*(|A\cup B|-|A\cap B|)}$ (2).

*Gower-Legendre (GOW)* is defined in [4] with formula: $GOW(A,B) = \frac{|A\cap B|}{|A\cap B|+\frac{1}{2}*(|A\cup B|-|A\cap B|)}$ (3).

*Sokal-Sneath (SOK)* is the last one in Jaccard family with formula [4]: $SOK(A,B) = \frac{|A\cap B|}{|A\cap B|+2*(|A\cup B|-|A\cap B|)}$ (4).

Notice that GOW and SOK are two variations of JAC and the difference between JAC, GOW and SOK is on the weight that each measure puts on the difference between input sets (i.e., |A ∪ B| − |A∩B|). For the same inputs, similarity values are higher/lower for GOW/SOK than JAC. For example, taking *sn1* and *sn2* as input, |sn1 ∪ sn2| = 6, and |sn1 ∩ sn2| = 3. Therefore, JAC(sn1, sn2) = 3/6 = 0.5, GOW(sn1, sn2) = 6/9 = 0.6667, and SOK(sn1, sn2) =1/3= 0.3333. All these set-based similarity algorithms actually normalize their similarity values based on the number of elements in the intersection set of two input sets and the number of different elements.

#### 2.2.2 Sequence-Based Similarity Measures

For sequence-based similarity functions, the input sequences (use case scenarios) are taken as edit distance, which is defined as the minimum number of operations (insertions, deletions and substitutions) [6] and the order of elements in the input sequences matters. One well known sequence-based similarity function is Hamming Distance [16]. However, it is limited to identical length input sequences, which does not fit our purpose. We used four sequence-based similarity functions: LCS [6], LEV [6], NW [7] and SW [7].

*Normalized Longest Common Subsequence (NLCS).* Longest Common Subsequence (LCS) [6] is a classical computer science problem and has applications in bioinformatics. In order to make LCS be capable of calculating similarity value, we extend is as Normalized Longest Common Subsequence (NLCS): the length of LCS divided by the average length of input sequences. The formula is defined as:

$$NLCS_{A,B} = LCS_{A,B} / \left(\frac{lengthOf(A) + lengthOf(B)}{2}\right) \quad (5)$$

Where *lengthOf()* aims to calculate the number of steps contained in a scenario and $LCS_{A,B}$ is the length of LCS calculated by following formula:

$$\mathrm{LCS}'_{A,B}(i,j) = \begin{cases} 0, & if\ \ i = 0\ or\ j = 0 \\ \mathrm{LCS}'_{A,B}(i-1,j-1)+1, & if\ A_i = B_j \\ \max(\mathrm{LCS}'_{A,B}(i-1,j), \mathrm{LCS}'_{A,B}(i,j-1)), & if\ A_i \neq B_j \end{cases} \quad (6)$$

Where $A_i$ is the $i_{th}$ step of scenario A, $B_j$ is the $j_{th}$ step of scenario B, and *i* and *j* are the lengths of scenarios A and B, respectively. Let us take *sn1* and *sn2* as input, *LCS(sn1, sn2)*=3 with the longest common subsequence: "*s1, s2, s6*.", therefore, *NLCS(sn1, sn2)*=3/((6+4)/2)=0.6.

*Levenshtein Distance (LEV).* LEV [6] is a well-known algorithm implementing edit-distance. In [6], the Levenshtein distance is defined as each mismatch (substitutions) or gap (insertion/deletion) increases the distance by one unit. To change distances into similarities, the function rewards each match and penalizes each mismatch and gap. In our current implementation, we use the basic setting: mismatch and gap are penalized the same by giving one point to increase the diversity (distance) and matches are given no penalty or reward. The Levenshtein distance between two scenarios A and B is defined as:

$$\underset{i\le m, j\le n}{LEV_{A,B}(i,j)} = \begin{cases} \max(i,j), \; if \min(i,j) = 0 \\ min \begin{cases} 1 + \mathrm{LEV}_{A,B}(i-1,j) \\ 1 + \mathrm{LEV}_{A,B}(i,j-1) \\ 1_{A_i \neq B_j} + \mathrm{LEV}_{A,B}(i-1,j-1) \end{cases} \end{cases} \quad (7)$$

When taking *sn1* and *sn2* as inputs, the first two elements in *sn1* and *sn2* match, and there is one mismatch ("*sn3, sn4*" or "*sn3, sn5*") combined with two gaps ("*sn5, sn3*" or "*sn3, sn3*") and one match ("*sn6, sn6*"). Therefore the Levenshtein distance is 3 meaning that the minimum edit operations required to change *sn1*/*sn2* into *sn2/sn1* is 3.

*Global alignment and Needleman-Wunsch (NW)*. An alignment of two sequences is a mapping between positions of their elements [7]. The goal of an alignment algorithm is to find the best way of positioning the elements of input sequences to maximize the alignment score. An alignment score measures matches, mismatches and gaps. Which is actually a similarity value. Global alignment is an algorithm that aligns the entire input sequences. The most basic global alignment algorithm is Needleman-Wunsch [7], and we use match score +1, mismatch 0 and a gap (Deletion, Insertion) of 0 as the operation weights of this similarity function. The scoring matrix *F* for NW is defined as below:

$$\underset{1\le r\le i}{F_{[r][0]}} = r * d, \underset{1\le c\le j}{F_{[0][c]}} = c * d;$$

$$\underset{1\le r\le j, 1\le c\le j}{F_{[r][c]}} = max \begin{cases} F_{[r-1][c-1]} + sim(A_r, B_c) \\ F_{[r-1][c]} + d \\ F_{[r][c-1]} + d \end{cases} \quad (8)$$

Where A and B are the input sequences, function *sim(A<sub>r</sub>, B<sub>c</sub>)* returns the match/mismatch scores between the $r_{th}$ member of *A* and the $c_{th}$ member of *B*, and *d* is the gap penalty. The similarity between *A* and *B* is $F_{[i][j]}$ where *i* and *j* are the lengths of *A* and *B* respectively. Therefore the similarity function is defined as:

$$NW_{A,B}(i,j) = F_{[i][j]} \quad (9)$$

Similarly, let us take sn1 and sn2 as input, then NW(sn1, sn2)=3.

*Local alignment and Smith-Waterman (SW)*. The goal of local alignment is to find the best alignment for sub-sequences of two input sequences. The output of a local alignment similarity function is two aligned substrings with the highest alignment score. SW is a commonly applied local alignment algorithm [7], where the scoring matrix *F* is defined in a similar way as the NW scoring matrix:

$$\underset{1\le r\le i}{F_{[r][0]}} = 0, \underset{1\le c\le j}{F_{[0][c]}} = 0;$$

$$\underset{1\le r\le j, 1\le c\le j}{F_{[r][c]}} = max \begin{cases} F_{[r-1][c-1]} + sim(A_r, B_c) \\ F_{[r-1][c]} + d \\ F_{[r][c-1]} + d \end{cases} \quad (10)$$

Note that each single element should not be negative, and the final similarity value is the biggest $F_{[r][c]}$, which indicates the most similar subsequence of the two input sequences. Therefore, the similarity function is defined as:

$$SW_{\boldsymbol{A,B}}(i,j) = \max_{1\le r\le i, 1\le c\le j}(F_{[r][c]}) \quad (11)$$

Where *i* and *j* are the lengths of the two input sequences A and B, respectively, and *F* is the scoring matrix defined above. Take *sn1* and *sn2* as input, then *SW(sn1, sn2)*=3.

## 3. PROBLEM REPRESENTATION AND FITNESS FUNCTIONS

Our objective is to select a most diverse subset of use case scenarios from the generated ones to enable cost-effective, manual requirements inspection, and the selection is optimized with respect to their pairwise similarity.

### 3.1 Problem Representation

As described in Section 2, a use case generates a set of scenarios *Scen = {S<sub>1</sub>, S<sub>2</sub>, S3 …. S<sub>n</sub>}*. Different scenarios may contain common steps; therefore each pair of the scenarios ($S_i$, $S_j$) can be assigned with a similarity value indicating their commonality. We summarize our problem as follows. We first encode the solution as a binary string $\langle 0|1|0|\cdots|1|0|1\rangle$ and a bit value "1" indicates the corresponding scenario is selected in the current solution. We then use the eight similarity functions to calculate the average similarity of scenarios obtained from the previous step. At last, after 2000 evaluations, the subset of scenarios with smallest similarity value is returned as the final solution. As we want to diversify selected scenarios, smaller similarity values are preferable.

In theory, the number of scenarios of a candidate subset ranges from 2 to *n* and the search space is the number of possible combinations of the number of scenarios for a given use case. For example, if the number of scenarios of a use case is *n*, then the overall space is $C_2^n + C_3^n + C_4^n + ...C_n^n = 2^n - n - 1$. Therefore, given a set of generated scenarios for a use case ($s_n$) and a particular similarity function (*SimFunc*), the selection of diverse use case scenarios can be formulated as *Minsr*($s_n$):

$$Minsr(Sn) = \min_{1\le p\le 2^n-n-1}\big(Similarity(Sm)\big) \quad (12)$$

$$\underset{2\le m\le n}{Similarity(Sm)} = \frac{\sum_{(S_i,S_j\in Sm \wedge i<j)} SimFunc(S_i,\, S_j)}{C_2^m} \quad (13)$$

Where *m* is the number of scenarios of the selected subset $s_m$, *SimFunc(*$s_i$*, *$s_j$*)* returns the similarity of two scenarios in $s_m$, and *n* is the total number of scenarios generated from use case $s_n$. Depending on which similarity function to apply, *SimFunc(*$s_i$*, *$s_j$*)* can be *CNT(*$s_i$*, *$s_j$*)*, *JAC (*$s_i$*, *$s_j$*)*, *GOW(*$s_i$*, *$s_j$*)*, *SOK(*$s_i$*, *$s_j$*)*, *NLCS(*$s_i$*, *$s_j$*)*, *LEV (*$s_i$*, *$s_j$*)*, *NW (*$s_i$*, *$s_j$*)*, or *SW(*$s_i$*, *$s_j$*)* (Section 2.2).

### 3.2 Fitness Function

For similarity functions *CNT*, *JAC*, *GOW*, *SOK*, *NW*, *SW* and *NLCS*, a lower value means more diversity. We define the fitness functions as:

$$\mathrm{Fitness}(Sm) = \frac{\sum_{i=1}^{m-1}\sum_{j=i+1}^{m} Nor(SimFunc(S_i,\, S_j))}{C_2^m} \quad (14)$$

A fitness value calculated by the above fitness functions ranges from 0 to 1, where a value closer to 0 means that the selected scenarios are different. In case of *LEV*, a higher value means more diversity and thus we define the fitness function as:

$$\mathrm{Fitness}_{LEV}(Sm) = \frac{\sum_{i=1}^{m-1}\sum_{j=i+1}^{m}(1.0 - Nor(SimFunc(S_i,\, S_j)))}{C_2^m} \quad (15)$$

In all the fitness functions, *m* is the number of the selected scenarios (subset *Sm*). Since different similarity functions produce different ranges of values, we normalize them between 0 and 1 using the normalization function [1]: $Nor(x) = \frac{x}{x+1}$, where *x* is a similarity value computed by a particular similarity function.

## 4. EMPIRICAL EVALUATION

In Section 4.1, we describe the experiment design. Section 4.2 presents statistical tests applied in the analysis. The experiment execution is presented in Section 4.3. Results are provided in Section 4.4 and a summary is presented in Section 4.5.

## 4.1 Experiment Design

This section describes our experiment design, including case studies, research questions, parameter settings of selected search algorithms and artificial problems.

### *4.1.1 Case Studies*

Our evaluation is based on one industrial case study and six other case studies from the literature. Their characteristics are summarized in Table 1. In total, we obtained 38 use cases, each of which forms a particular optimization problem. In Table 1, we also reported the total number of scenarios that are automatically derived for each case study based on the *All Condition Coverage* criterion (Section 2.1). The average length (column *Avg_Len*) of the scenarios for each case study is also reported in Table 1.

The industrial case study is a *Navigation System* (NAS), which controls and guides the system based on control law computation that takes data sampled from sensors as input and sends commands to actuators. The other six case studies were derived from the literature. ATM is from [9]. Crisis Management Systems (CMS) was originally presented in [11]. We modeled one of its key use case named as *Communicate with other coordinator*. Payroll, CPD, VS and NGP are course materials. Notice that some of these case studies have been used to in [15] to evaluate RUCM.

**Table 1. Characteristics of the case studies**

| Category | Case studies | # Use Cases | #Scenarios | Avg_Len |
|---|---|---|---|---|
| Industrial Case Study | Navigation System (NAS) | 11 | 792 | 23 |
| From the Literature | Bank System (ATM) | 4 | 221 | 20 |
| | Crisis Management System (CMS) | 1 | 282 | 27 |
| | Car Part Dealer (CPD) | 6 | 122 | 29 |
| | Video System (VS) | 7 | 53 | 22 |
| | Next Generation POS (NGP) | 1 | 34 | 14 |
| | Payroll System (PAY) | 8 | 163 | 27 |
| | **SUM** | **38** | **1667** | **23.2** |

### *4.1.2 Research Questions*

Our experiments aim to evaluate the proposed fitness functions together with the selected search algorithms and the similarity functions in terms of optimally selecting a subset of scenarios by maximizing their diversity. With respect to this objective, we would like to answer the following research questions: ***RQ1***: Are the search algorithms effective to solve our optimization problem to compare with RS? ***RQ2***: Among the four search algorithms, which one fares best in solving our optimization problem? ***RQ3***: Among the eight similarity functions, which one performs best for optimizing our problem? ***RQ4***: How does the combination of a similarity function and search algorithm impact the results?

### *4.1.3 Parameter Settings for Search Algorithms*

We compared four algorithms: AVM, SSGA, (1+1) EA and RS. AVM was selected as a representative local search algorithm and each variable can either be "0" or "1" in our problem. SSGA was selected since it is the most commonly used global search algorithm [8]. Population size of SSGA was set to 100; crossover rate was set to 0.75; whereas 1.5 bias was used for rank selection. The standard 1-point crossover operator was employed with mutation rate of 1/*nvar*, where *nvar* represents the total number of variables. In case of (1+1) EA, the population size is set to one. Finally, for the sanity check, RS is used as the baseline.

## 4.2 Statistical Tests

Following the guidelines on the use of appropriate statistical tests for search algorithms [3], we used the Vargha and Delaney statistics [24] and Mann–Whitney U test (also called the Wilcoxon rank-sum test). The Vargha and Delaney statistics is an effect size that computes $\hat{A}_{12}$. Notice that it is a non-parametric measure, i.e., it doesn't assume the normality of the sample. If $\hat{A}_{12}$ is 0.5, this means two algorithms have equal performance; a value greater than 0.5 suggests the first algorithm has higher chance to obtain better solutions than B, and a value less than 0.5 implies vice-versa. The Mann–Whitney U test computes $p$-value for deciding if there is a significant difference between two algorithms. We choose the significance level of 0.05.

## 4.3 Experiment Execution

In our experiments, we run each algorithm 100 times to deal with random variations and the number of generations was set to 2000. We collected the final optimal solutions for evaluation of 2000th generation. We ran our experiments on Abel computer cluster with eight computing nodes, each of which has 16 physical computer cores and 64 GB of memory.

## 4.4 Results and Analyses

### *4.4.1 RQ1 and RQ2*

To answer RQ1 and RQ2, we conducted the Vargha and Delaney statistic test and Mann–Whitney U test. Results for RQ1 show that AVM, (1+1) EA and SSGA significantly outperformed RS for most of the problems regardless which similarity function was applied. To answer RQ2, we compared each algorithm pair and results are presented in Table 2. The column A>B is the number of problems (out of 38) that A is significantly better than B; A<B means vice versa; and A=B means the number of problems for which there no significant differences since $p$-value>=0.05.

**Table 2. Results for RQ2 using Vargha and Delaney statistics and Mann–Whitney U test at significance level of 0.05**

| Similarity Function | Pair of Algorithms | A > B | A < B | A = B |
|---|---|---|---|---|
| CNT | AVM vs. SSGA | 31 | 5 | 2 |
| | (1+1) EA vs. SSGA | 36 | 1 | 1 |
| | AVM vs. (1+1) EA | 2 | 34 | 2 |
| JAC | AVM vs. SSGA | 30 | 7 | 1 |
| | (1+1) EA vs. SSGA | 36 | 2 | 0 |
| | AVM vs. (1+1) EA | 1 | 36 | 1 |
| GOW | AVM vs. SSGA | 29 | 5 | 4 |
| | (1+1) EA vs. SSGA | 37 | 1 | 0 |
| | AVM vs. (1+1) EA | 0 | 38 | 0 |
| SOK | AVM vs. SSGA | 29 | 7 | 2 |
| | (1+1) EA vs. SSGA | 35 | 2 | 1 |
| | AVM vs. (1+1) EA | 0 | 38 | 0 |
| NLCS | AVM vs. SSGA | 31 | 6 | 1 |
| | (1+1) EA vs. SSGA | 36 | 2 | 0 |
| | AVM vs. (1+1) EA | 0 | 37 | 1 |
| LEV | AVM vs. SSGA | 29 | 6 | 3 |
| | (1+1) EA vs. SSGA | 35 | 1 | 2 |
| | AVM vs. (1+1) EA | 0 | 38 | 0 |
| NW | AVM vs. SSGA | 32 | 5 | 1 |
| | (1+1) EA vs. SSGA | 34 | 0 | 4 |
| | AVM vs. (1+1) EA | 1 | 37 | 0 |
| SW | AVM vs. SSGA | 32 | 6 | 0 |
| | (1+1) EA vs. SSGA | 33 | 0 | 5 |
| | AVM vs. (1+1) EA | 1 | 37 | 0 |

In summary, ***AVM vs. (1+1) EA:*** For CNT, (1+1) EA significantly outperformed AVM for 36 problems; AVM significantly outperformed (1+1) EA for only one problem; and there was no significant difference for one problem. Similar cases can be observed for the other similarity functions. ***(1+1) EA vs. SSGA:*** For most of the problems, (1+1) EA performed significantly better than SSGA. Taking JAC for example, (1+1) EA significantly performed SSGA for 36 problems, while SSGA significantly outperformed (1+1) EA for only two problems. ***AVM vs. SSGA:*** AVM performed significantly better than SSGA for most of the problems. Taking example of JAC, AVM performed significantly better for 30 problems; SSGA performed significantly better for seven problems; there was no difference for one problem. We therefore can conclude that for RQ2, (1+1) EA significantly outperformed AVM and AVM significantly outperformed SSGA for most of the problems regardless which similarity function was applied.

### *4.4.2 RQ3*

RQ3 aims to answer which similarity function performs significantly better. To address RQ3, similar to RQ1 and RQ2, we performed the Vargha and Delaney statistics and Mann–Whitney U test to evaluate the differences of the eight similarity functions.

In order to perform these statistical tests between two similarity functions, we re-evaluated similarity function values of the obtained solution (i.e., a set of scenarios) of one similarity function, using the similarity function of the other one. Doing so is to make sure that two samples of two similarity functions are comparable. For example, when comparing CNT and JAC in terms of AVM, we use the similarity function of JAC to re-evaluate the optimal solution obtained by the combination of CNT and AVM, and perform the statistical tests. The comparison is both ways, in the sense that we use the similarity function of CNT to re-evaluate the optimal solution obtained by the combination of JAC and AVM and perform corresponding statistical tests.

We first compare the performance of set-based and sequence-

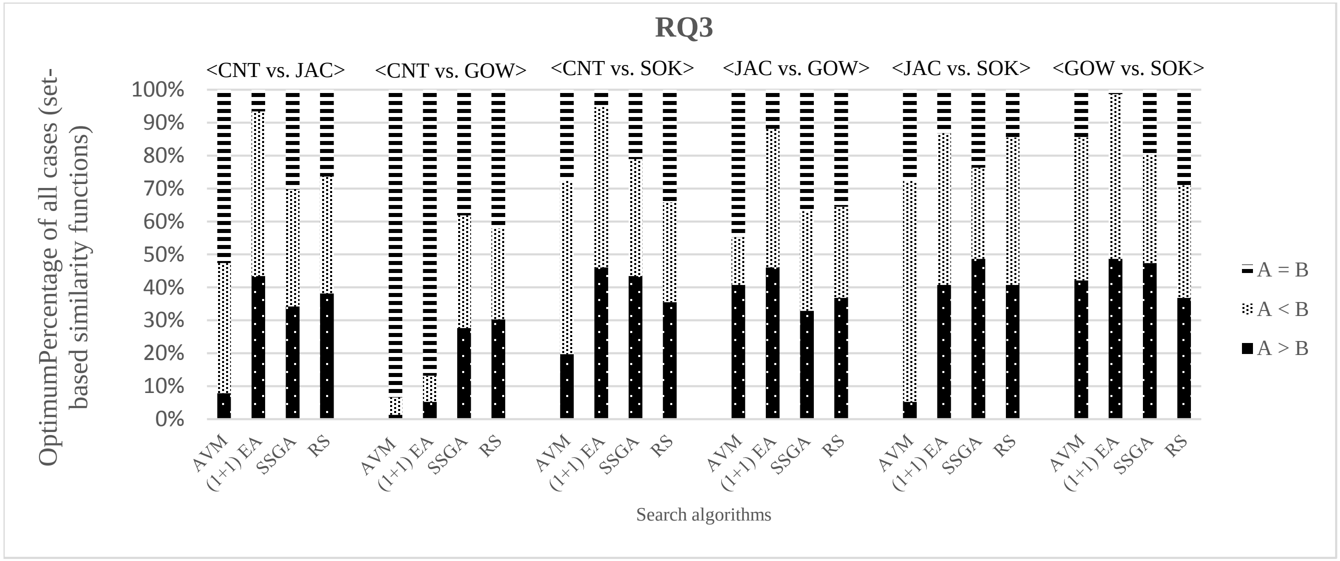

**Fig. 1. OptimumPercentage of all set-based similarity functions**

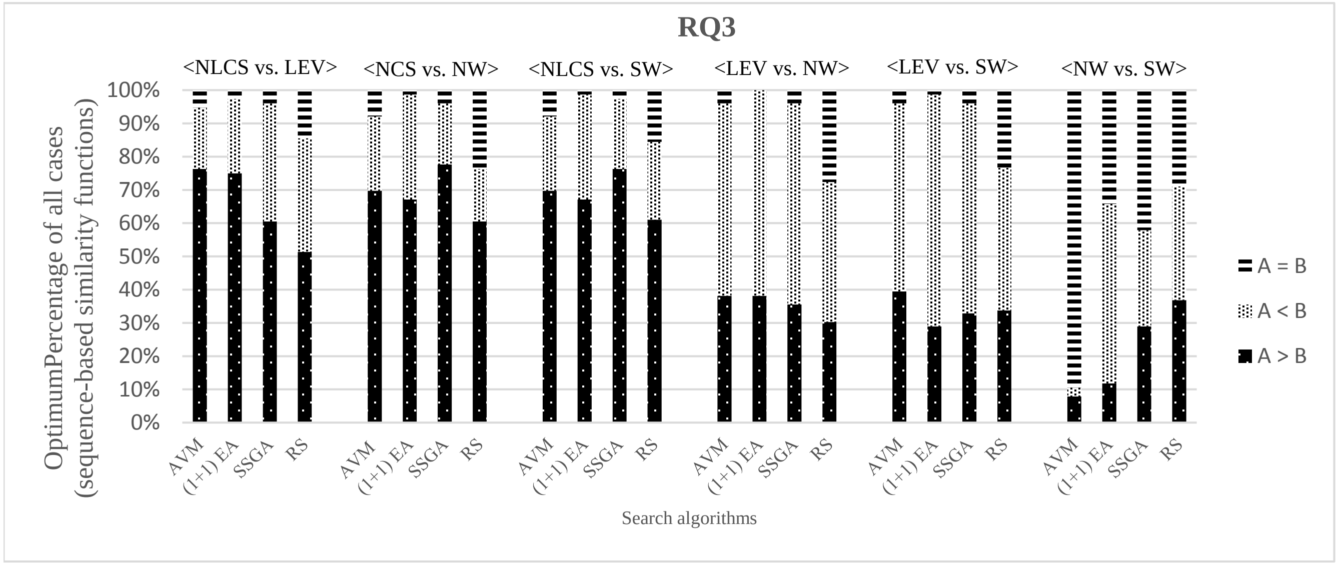

**Fig. 2. OptimumPercentage of all sequence-based similarity functions**

based measures separately and identify the best function of each class. We then compare the best set-based one with the best sequence-based one. Based on the data collected from the experiment, we conclude that NLCS is the best among all the sequence-based measures. It is however not easy to conclude which set-based similarity function is the best one. Detailed experiments data regarding the performance of set-based and sequence-based measures are reported in the Fig. 1 and Fig. 2, respectively. Therefore, we compared NLCS with all the four set-based similarity functions. To enhance the comprehension of these results, we compared NLCS with each set-based similarity function for solving each problem by counting the number of problems for each algorithm where it performs the best in terms of fitness values. The percentage of the problems that a combination is the best among the all is named as OptimumPercentage and the both-ways evaluation and statistical tests results are eventually reported in Fig. 3. For example, in context of NLCS vs. JAC as shown in Fig. 3, for AVM, NLCS significantly outperformed JAC for 60.53% of all cases (total number of the evaluation results), as the black bar (A > B) indicating that one performed significantly better than the other. Notice that the total number of the evaluation results for each combination of a search algorithm and a similarity function are then doubled the total number of the problems (i.e., 38).

We summarize the key results below. ***NLCS vs. CNT:*** For AVM, NLCS achieved the best *OptimumPercentage* (51.32%), implying that NLCS significantly outperformed CNT for more than half of the cases while CNT only performed significantly better than NLCS for 28.94% of all cases. Similar results can be observed for other search algorithms, for (1+1) EA, NLCS significantly outperformed CNT for 65.79% of all cases. For SSGA, NLCS achieved the best *OptimumPercentage* (64.47%). For RS, NLCS performed significantly better than CNT for 48.68% of all cases while CNT only achieved the OptimumPercentage of 27.64% (Fig. 3). ***NLCS vs. JAC:*** For AVM, NLCS significantly outperformed JAC for 60.53% of all cases, while JAC only achieved the *OptimumPercentage* of 28.95%. Similar results can be observed for other search algorithms, we therefore can draw the conclusion that NLCS is significantly better than JAC. ***NLCS vs. GOW:*** For (1+1) EA, NLCS achieved the best *OptimumPercentage* (64%) while GOW only significantly outperformed NLCS for 28% of all cases. For other search algorithms, NLCS always achieved the best *OptimumPercentage* as shown in Fig. 3. ***NLCS vs. SOK:*** NLCS always achieved the best *OptimumPercentage* no matter which search algorithm is applied. Taking SSGA for example, NLCS significantly outperformed SOK for 63.16% of all cases, and SOK performed significantly better than NLCS only for 28.95% of all cases.

Therefore, based on the data presented in Fig. 3, we can then conclude that NLCS significantly outperformed all the other set-based similarity functions, no matter which search algorithm is applied. Thus NLCS is preferable for solving our problems.

### *4.4.3 RQ4*

To address RQ4, we evaluate the obtained optimal solution from each combination of the four search algorithms and eight similarity functions. Results are reported in Table 3.

**Table 3. ReliabilityPercentage Values of Each Combination of the Similarity Functions and Search Algorithms**

| Similarity Measures | Search Algorithms | | | |
|---|---|---|---|---|
| | **AVM** | **(1+1) EA** | **SSGA** | **RS** |
| CNT | 75.92% | 76.97% | 55.06% | 21.01% |
| JAC | 63.68% | 75.19% | 58.51% | 19.34% |
| GOW | 75.85% | 79.83% | 65.26% | 21.51% |
| SOK | 63.55% | 64.37% | 61.11% | 16.43% |
| NLCS | 79.67% | **87.59%** | 65.92% | 23.46% |
| LEV | 41.31% | 42.26% | 39.93% | 11.97% |
| NW | 44.47% | 46.44% | 40.46% | 13.88% |
| SW | 42.46% | 45.82% | 40.75% | 13.68% |

For each problem we obtain 32 solutions and each solution is a set of use case scenarios, and we pick up scenarios shared among all the solutions into a set (*Best-Set*). We then report the value of *ReliabilityPercentage* (one cell in Table 3) representing the probability of a solution to be the *Best-Set*, i.e., how many scenarios of a solution are selected into *Best-Set*. For example, value 75.92% in the 3rd row and 2nd column shows that 75.92% of the scenarios of the solutions obtained by the combination of AVM and CNT were shared with other combinations. Notice that, if two solutions with the same fitness value are total different, we then put both of them into the *Best-Set*.

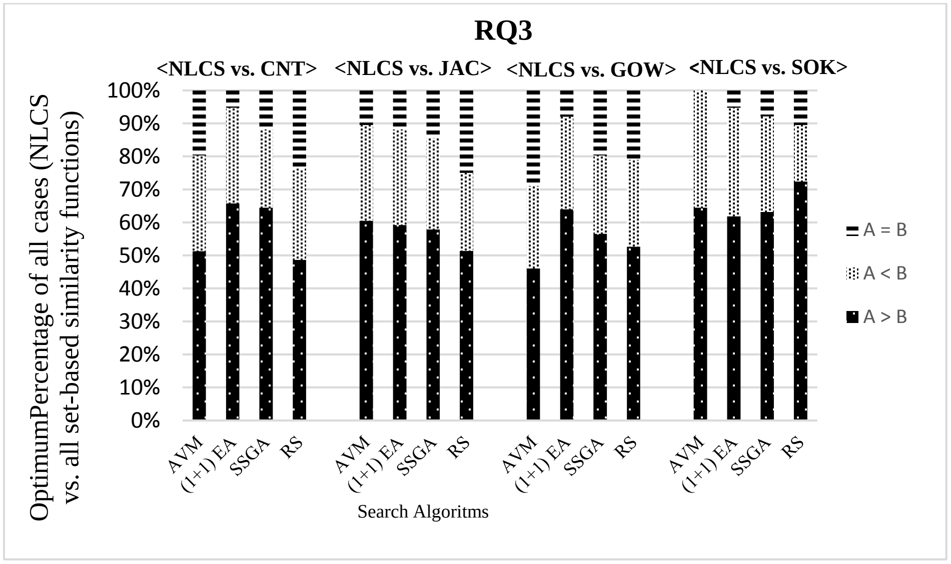

**Fig. 3. NLCS vs. all set-based similarity functions**

From the data presented in Table 3, we can observe that the combination of NLCS and (1+1) EA achieved the best *ReliabilityPercentage* (87.59%), indicating that the combination of NLCS and (1+1) EA obtained a high reliability 87.59% for solving our problems.

## 4.5 Summary and Overall Discussion

We observed that the performance of (1+1) EA, AVM and SSGA was significantly better than RS (RQ1), indicating that the fitness function is effective to guide search algorithms to solve our optimization problem. Among all the search algorithms, (1+1) EA was significantly better than the others (RQ2), followed by AVM and then SSGA. This is because (1+1) EA is a global search algorithm and manages to find global optimal solutions as compared to AVM (a local search algorithm). Though SSGA is also a global search algorithm, it relies on both mutation and crossover operators for exploration and exploitation of the search space. Therefore, SSGA might need more generations to exploit (via the crossover operator) on a specific area of the search space in order to obtain an optimal solution. In contrast, (1+1) EA uses only the mutation operator to explore the search space and hence manages to find optimal solutions quicker than SSGA.

NLCS significantly outperformed all the others for most of the cases when combined with any search algorithm (RQ3). For set-based ones, CNT and GOW performed similarly for most of the cases. One possible reason is that, in the current implementation of the *Loop Coverage* criterion for deriving use case scenarios, each loop is exercised exactly once. Therefore there are not many repeated steps (sentences) in automatically generated use case scenarios, which makes the effect of CNT and GOW very minor. One can expect that a different implementation of the *Loop Coverage* criterion might lead to a different result. We plan to further investigate it in the future. For the sequence-based similarity functions, NLCS performed significantly better than the others. It may be because NLCS normalizes the length of LCS with respect to the average length of two input sequences, which makes the similarity value more precise as compared to the other sequence-based similarity functions. To compare with the set-based ones, NLCS is more effective, because the order of steps of input scenarios indeed matters in our context.

## 5. THREATS TO VALIDITY

To reduce ***construct*** validity threats, we chose the effectiveness measure (fitness value) and the same stopping criterion (number of generations) for all the search algorithms. To reduce ***conclusion*** validity threats, we followed a rigorous statistical procedure to analyze collected data: the Vargha and Delaney statistics, Mann–Whitney U test (also called Wilcoxon rank-sum test) and Kruskal-Wallis test. Furthermore, we repeated experiments 100 times to tackle the random variation. A possible threat to internal ***validity*** is that we have experimented with only one configuration setting for the GA parameters. However, these settings conform to the common guidelines in the literature. One common ***external*** validity threat is about the generalization of results. We ran our experiments on one industrial case study and six case studies from the literature. However, to build further confidence on the results, more studies will be conducted in the future.

## 6. RELATED WORK

### 6.1 Search-Based Requirements Engineering

A survey of search-based software engineering is reported in [2]. This survey discusses various applications of search algorithms to various phases of software development, for instance, requirement, design, and testing. From all of these works, the works reporting requirements related applications are related to us and includes requirements selection and optimization, requirements assignment and requirements prioritization. Using search algorithms together with similarity functions to address optimization problems have already been applied in supporting testing. For example, in [12], the authors adopted similarity functions in conjunction with a classification algorithm to select test cases.

The approach in [23] takes requirements interaction management into the consideration of the automated requirements selection process and introduces five requirements dependencies (i.e., *And*, *Or*, *Precedence*, *Value-related*, *Cost-related*). Results of their empirical studies show that the *And* dependency appears to denote a tighter constraint than the *Or* and *Precedence* dependencies, and *Value-related* and *Cost-related* dependencies directly contribute to an increase or decrease in fitness values. Our approach however focuses on well-defined UML use case relationships (e.g., *Include*, *Extend*), which are actually requirements dependencies.

To the best of our knowledge, there is no work focusing on using similarity-based techniques combined with search algorithms to address the use case scenarios selection problem.

### 6.2 Inspection of Use Case Models

In software engineering, identifying defects in requirements is one of the most effective and efficient quality assurance techniques [18]. The authors of [10] proposed a tentative taxonomy of defects in use case models in conjunction with a checklist-based inspection technique to detect defects and the proposed checklist was evaluated in a controlled experiment. We carefully studied their checklist and found some items of the checklist have already been fulfilled in our approach. In fact, the RUCM built-in restriction rules help users to avoid such defects. Pete McBreen [20] proposed a checklist with 29 specific elements to conduct use case inspection that covers stakeholders, use case goal, use case structure, use case syntax, etc. In paper [21], the authors adopted usage-based reading technique to conduct use case inspection. An empirical study has been conducted to compare inspection techniques for detecting defects in use case descriptions [17]. The authors performed a systematic review of use case checklist techniques and proposed their checklist technique. Results show that the checklist found more defects than the ad hoc approach and groups found more defects than individuals.

Different from the above methods, our approach adopts search algorithms in conjunction with similarity measures to select most diverse use case scenarios to facilitate use case inspections. Notice that, rather than focusing on specific use case inspection techniques, our approach aims to pick up most diverse use case scenarios that deserve more attention and a higher priority during the inspection process.

### 6.3 Use Case Prioritization

An approach of use case scenario prioritization has been proposed in paper [9] to identify the criticality of a scenario by relying on the density of overlapping of the sub paths of a scenario path with other scenario path(s) of the same use case. We however optimally select diverse use case scenarios based on similarity measures.

## 7. CONCLUSION

Due to the inherent complexity of large-scale systems, a large portion of effort is allocated to requirements engineering as the quality of requirements has a significant impact on almost every single downstream activity of the development lifecycle of such a system. A large number of use cases often lead to a large number of use case scenarios for any non-trivial system. Use case scenarios are usually the input for manual requirements inspection and analysis. Finding a subset of use case scenarios to support the cost-effective manual inspection is therefore an important optimization problem to address. In this paper, we propose an innovative approach to address this optimization problem by combining similarity measures with search algorithms.

The proposed approach has been evaluated with an industrial case study and six case studies from the literature. Results show that all search algorithms significantly outperformed random search and among the algorithms (1+1) EA is the best. NLCS performed significantly better than the other similarity measures. The combination of NLCS with (1+1) EA gives the best results.

## 8. ACKNOWLEDGMENTS

This work is mainly supported by the Zen-Configurator project (project number: 240024) funded by the Research Council of Norway under the category of Young Research Talents of the FRIPO funding scheme. Huihui Zhang is supported by the exchange program between the Research Council of Norway and China Scholarship Council (project number: 249086). Chao Liu is supported by Ministry of Industry and Information Technology, China (project number: MJ-S-2012-05). We acknowledge the Abel computer cluster for supporting the conducting of the experiments.

## 9. REFERENCES

[1] Arcuri, A. 2013. It really does matter how you normalize the branch distance in search-based software testing. *Software Testing, Verification and Reliability*, 23, 2 (March 2013), 119-147.

[2] Harman, M., Mansouri, S. A. and Zhang, Y. 2009. *Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications*. Technical Report TR-09-03, King College London.

[3] Arcuri. A. and Briand, L. C. 2011. A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering. In *Proceedings of ICSE 2011*, 21-28.

[4] Xu, R. and Wunsch, D. 2005. Survey of clustering algorithms. *IEEE Trans on Neural Networks*, vol. 16 (2005), 645–678.

[5] Cartaxo, E. G., Machado, P. D. L. and Neto, F. G. O. 2011. On the use of a similarity function for test case selection in the context of model-based testing. *Software Testing, Verification and Reliability*, 21, 2 (2011), 75–100.

[6] Gusfield, D. 1997. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*, Cambridge University Press.

[7] Durbin, R., Eddy, S. R., Krogh, A. and Mitchison, G. 1999. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press.

[8] Ali, S., Briand, L. C., Hemmati, H. and Panesar-Walawege, R. K. 2009. A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation, *IEEE Trans on Software Engineering*, 36, 6 (August 2009), 742-762.

[9] Kundu, D. and Samanta, D. 2007. A Novel Approach of Prioritizing Use Case Scenarios, In *Proceedings of APSEC 2007* (Dec. 4-7, 2007), 542-549.

[10] Anda, B. and Sjøberg, D.I.K. 2002. Towards an Inspection Technique for Use Case Models. In Proceedings of *SEKE '02* (2002), 127-134.

[11] Capozucca, A., Cheng, B., Georg, G., Guelfi, N., Istoan, P. and Mussbacher, G. 2011. *Requirements Definition Document for a Software Product Line of Car Crash Management Systems*, Technical Report CS-11-105, 2011.

[12] Simao, A. D. S. Mello, R. F. D. and Senger, L. J. 2006. A technique to reduce the test case suites for regression testing based on a self-organizing neural network architecture. In *Proceedings of COMPSAC'06* (2006), 93–96.

[13] Ledru, Y., Petrenko, A. and Boroday, S. 2009. Using string distances for test case prioritization. *In proceedings of ASE 2009* (2009), 510–514.

[14] Pohl, K. 2010. *Requirements Engineering: Fundamentals, Principles, and Techniques*, Springer-Verlag Berlin and Heidelberg GmbH & Co. K. 2010.

[15] Yue, T., Briand, L. C. and Labiche, Y. 2013. Facilitating the transition from use case models to analysis models: Approach and experiments. *ACM Tranasction on Software Engineering and Methodology*, 22, 1(February 2013) Article 5.

[16] Dong G. and Pei, J. 2007. *Sequence Data Mining*, Springer-Verlag New York Inc., 2007.

[17] Cox, K., Aurum, A. and Jeffery, R. An Experiment in Inspecting the Quality of Use Case Descriptions, *Jornal of Research and Practice in Informaiton Technology*, vol. 36 (November 2004), 211-229.

[18] Porter, A., Votta, L. and Basili, V. 1995. Comparing detection methods for software requirements inspections: A replicated experiment, *IEEE Transactions on Software Engineering*, 21, 6 (1995), 563–575.

[19] Yue, T., Ali, S. and Zhang, M. Applying A Restricted Natural Language Based Test Case Generation Approach in An Industrial Context. *ISSTA 2015(in press)*.

[20] Mcbreen, P. Use Case inspection checklist, http://www.mcbreen.ab.ca/papers/QAUseCases.html. (last accessed, June 2015)

[21] Thelin, T., Runeson, P. and Regnell, B. 2001. Useage-based reading—an experiment to guide reviewers with use cases, *Information and Software Technology*, 43, 15 (December 2001), 925-938.

[22] Droste, S., Jansen, T. and Wegener, I. 2002. On the analysis of (1+1) evolutionary algorithm. *Theoretical Computer Science*, vol. 276 (2002), 51-81.

[23] Zhang, Y., Harman, M. and Lim, S.L. 2013. Empirical evaluation of search based requirements interaction management. *Information and Software*. vol. 55 (2013), 126-152.

[24] Vargha, A. and Delaney, H. D. 2000. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. Journal of Educational and Behavioral Statistics, vol. 25 (2), June 20, 2000, 101-132.

[25] Yue, T., Briand, L.C. and Labiche, Y. aToucan: An Automated Framework to Derive UML Analysis Models from Use Case Models. *ACM Transactions on Software Engineering and Methodology.* 24, 3, (May, 2015), Article 13.

[26] Goldber, D.E. Genetic Algorithms in Search, Optimization and Machine Learning: Addison-Wesley Professional, 2001.

[27] Kempka, J., McMinn P. and Sudholt, D. A theoretical runtime and empirical analysis of different alternating variable searches for search-based testing. In *Proceedings of GECCO'13*, (July 06-10, 2013), 1445-1452.